1 Основные задачи

1. Решите следующие уравнения на лямбда-термы:
   (a) $\forall M.\, F\ M =_\beta M\ F$
   (b) $\forall M.\, M\ F =_\beta M\ (M\ F)$
   (c) $\forall M\ N.\, F\ M\ N =_\beta F\ N\ (M\ F)$
   (d) $\forall M.\, F\ M =_\beta F\ M$

2. Даны термы A и B. Найдите F и G такие, что
$$\begin{cases} F =_\beta A\ F\ G \\ G =_\beta B\ F\ G \end{cases}$$

3. Используя комбинатор примитивной рекурсии, напишите терм, который находит n-ое число Фибоначчи.

4. Находится ли данный терм в нормальной форме? В головной? В слабой головной? Почему?
$$\lambda y.\, z\ (\lambda x.\, \mathbf{I}\ (\mathbf{I}\ \mathbf{I})\ (\mathbf{I}\ \mathbf{I}))\ x\ y$$
   Дважды проредуцируйте этот терм до нормальной формы, расписывая каждый шаг редукции: один раз с использованием аппликативной стратегии, во второй — с использованием нормальной стратегии.

5. Напишите функцию, которая сильно нормализуемо возвращает последний элемент непустого списка.

6. Напишите функцию **tail**, которая сильно нормализуемо возвращает хвост списка, то есть список без первого элемента: например, **tail** $[\,] = [\,]$ и **tail** $[5, 10, 13] = [10, 13]$.

7. - Докажите, что для каждого терма M существует терм N такой, что N находится в нормальной форме и $N\ \mathbf{I} =_\beta M$.
   - Задайте *сильно нормализуемый* (!) набор термов **hasValue**, **getValue**, **value** и **noValue** с такими правилами:
$$\begin{aligned} &\mathbf{hasValue}\ (\mathbf{value}\ a) = \mathbf{true} \\ &\mathbf{hasValue}\ \mathbf{noValue} = \mathbf{false} \\ &\mathbf{getValue}\ (\mathbf{value}\ a) = a \\ &\mathbf{getValue}\ \mathbf{noValue} = \bot \end{aligned}$$
     Здесь $\bot$ — произвольный терм на выбор решающего такой, что у него нет нормальной формы. Понадобится воспользоваться предыдущим пунктом.

8. Предположим, что у нас уже есть набор термов **hasValue**, **getValue**, **value** и **noValue**, на котором выполняются правила выше. Тогда дадим альтернативное определение списков:
$$\begin{aligned} &\mathbf{nil}' := \mathbf{noValue} \\ &\mathbf{cons}' := \lambda h\ t.\, \mathbf{value}\ (\mathbf{pair}\ h\ t) \end{aligned}$$
   - Постройте функцию, которая по такому определению возвращает список в формате, определённом на занятии. Например, должно выполняться
$$\mathbf{dataListToChurch}\ (\mathbf{cons}'\ a\ (\mathbf{cons}'\ b\ \mathbf{nil}')) = \mathbf{cons}\ a\ (\mathbf{cons}\ b\ \mathbf{nil})$$

- Постройте обратную функцию, которая по списку, заданному согласно определению с занятия, возвращает список в заданном виде.

$$\textbf{churchToDataList}\ (\textbf{cons}\ a\ (\textbf{cons}\ b\ \textbf{nil})) = \textbf{cons}'\ a\ (\textbf{cons}'\ b\ \textbf{nil}')$$

  Исполнение должно быть сильно нормализуемым.

9. Вспомним определение умножения чисел Чёрча:

$$\textbf{mult} = \lambda n\ m\ s.\, n\ (m\ s)$$

   Рассмотрим его аналог для списков:

$$\textbf{listMult} = \lambda n\ m\ s.\, n\ (\lambda e.\, m\ (\textbf{curry}\ s\ e))$$

   Что делает эта функция?

10. Известно два определения увеличения числа Чёрча на единицу:

$$\textbf{suc} := \lambda n\ s\ z.\, s\ (n\ s\ z)$$
$$\textbf{suc}' := \lambda n\ s\ z.\, n\ s\ (s\ z)$$

    На занятии мы рассмотрели, как по аналогии с **suc** можно построить функцию **cons** — терм, который дописывает в начало списка новый элемент.

    - Что получится по аналогии с $\textbf{suc}'$?
    - Задайте лямбда-терм, который принимает на вход список и сильно нормализуемо возвращает результат его разворота:

$$\textbf{reverse}\ [1, 3, 6, 7] = [7, 6, 3, 1]$$

2 Дополнительные задачи

Задания на случай нехватки баллов.

1. Придумайте и опишите свою стратегию редукции. Найдите терм, на котором аппликативная, нормальная и выдуманная стратегии редукции дают разные последовательности редуцирований, и опишите эти последовательности.

2. Из лекции известно, что если нормальная форма у лямбда-терма существует, то она единственная. Докажите, что не существует такого лямбда-терма F, что для любых термов N и M выполняется $F\ (M\ N) =_\beta M$. Для этого подберите такие термы M и N, что если бы такой F существовал, то у терма $F\ (M\ N)$ было бы несколько различных нормальных форм.

3. Реализуйте функцию, которая принимает на вход f и n и возвращает

$$\prod_{i=0}^{n-1} f(i)$$

   (Это обозначение для $f(0) \cdot f(1) \cdot f(2) \cdots f(n-1)$).

4. Напишите функцию `listrec`, которая эквивалентна такому коду и сильно похожа на комбинатор примитивной рекурсии для натуральных чисел:

```
def listrec(f, a, lst):
  cur = []
  acc = a
  for el in reversed(lst):
    cur = [el] + cur
    acc = f(acc, el, cur)
  return acc
```

Применение этой функции должно быть сильно нормализуемо.

5. Напишите функцию **tail′** для списков, реализованных через **value** (см. основные задачи).

6. Напишите **listrec′** для таких списков.

7. Представьте в кодировании по Чёрчу бинарное дерево

   ```
   data Tree a = Leaf | Node (Tree a) a (Tree a)
   ```

8. Даны двоичные числа с ведущими нулями в лямбда-исчислении (*закодированные по Чёрчу*):

   $$\begin{aligned} \mathbf{ZB} &:= \lambda\ f_0\ f_1\ z.\ z \\ \mathbf{IB} &:= \lambda\ n\ f_0\ f_1\ z.\ f_1\ (n\ f_0\ f_1\ z) \\ \mathbf{OB} &:= \lambda\ n\ f_0\ f_1\ z.\ f_0\ (n\ f_0\ f_1\ z) \end{aligned}$$

   ZB — конструктор нуля, **IB** — конструктор, принимающий n и возвращающий $2 \cdot n + 1$, **OB** — конструктор, принимающий n и возвращающий $2 \cdot n$. Таким образом, последовательность **IB** и **OB** кодирует биты некоторого числа.

   У этих чисел есть небольшая проблема: они поддерживают несколько по смыслу идентичных, но структурно различающихся способов записи каждого числа, поскольку **ZB**, **OB ZB**, **OB (OB ZB)** и так далее все могут кодировать число 0.

   Несколько примеров чисел в таком кодировании:

0	`0`	**ZB**
1	`1`	**IB ZB**
2	`10`	**OB (IB ZB)**
3	`11`	**IB (IB ZB)**
4	`100`	**OB (OB (IB ZB)**
5	`101`	**IB (OB (IB ZB)**
6	`110`	**OB (IB (IB ZB)**
003	`0011`	**IB (IB (OB (OB ZB)))**
0002	`00010`	**OB (IB (OB (OB (OB ZB))))**

   Напишите функцию, которая находит количество битов в таком числе, считая и ведущие нули.

9. Напишите функцию, которая находит количество единичных битов в двоичном числе.

10. Напишите функцию, которая находит количество битов в таком числе, не считая ведущие нули.

11. Напишите функцию, которая прибавляет единицу к двоичному числу.

12. Напишите функцию, которая переводит двоичное число в число Чёрча.

13. Напишите функцию, которая переводит число Чёрча в двоичное.

14. Напишите функцию, которая находит квадратный корень числа Чёрча...

15. ... без использования комбинатора неподвижной точки.
16. Напишите функцию, которая делит число Чёрча на два...
17. ... без использования комбинатора неподвижной точки.
18. Напишите функцию, которая делит одно число Чёрча на другое...
19. ... без использования комбинатора неподвижной точки.